Amendments to the Claims

The listing of claims will replace all prior versions, and listings of claims in the application.

1. (currently amended) A method of on-the-fly patching of executable code comprising:

identifying original instructions to be changed while the original instructions are being executed on a processor;

copying the original instructions to a storage location;

~~adding a jump instruction to the copied instructions to return to a next instruction after the original instructions~~ adding a hook with a first jump instruction for transferring control to the copied instructions;

using a second jump instruction in the copied instructions for transferring control to an unpatched instruction in a location where the instructions are being patched; and

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook,~~;

calling the hook;

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called;

wherein the original instructions are part of the instruction set of the processor available to a user, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions.

2. (previously presented) The method of claim 1, further comprising, prior to the copying step, allowing a write operation on a page in memory where the original code is located.

3. (previously presented) The method of claim 1, further comprising, prior to the copying step, masking interrupts.

4. (original) The method of claim 1, further comprising, after the replacing step, disallowing a write operation on the page in memory where the block of code is located.

5. (original) The method of claim 1, further comprising, after the replacing step, unmasking interrupts.

6. (original) The method of claim 1, wherein the original instructions are changed in reverse order.

7. (previously presented) The method of claim 1, wherein the mark instructions are the same length, in bytes, as the instructions to be patched.

8. (currently amended) The method of claim 1, wherein the mark instructions are shorter in length, in bytes, as the instructions to be patched, and ~~include~~ wherein NOP (no operation) filler follows the mark instructions.

9. (currently amended) A method of on-the-fly patching of executable code comprising:

identifying original instructions to be changed while the original instructions are being executed on a processor;

copying the original instructions to a storage location;

~~adding a jump instruction to the copied instructions to return to a next instruction after the original instructions~~ adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook,~~; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, and

wherein the modified instructions include a resolver to determine a number of the instructions at a location of the original code that had already been executed.

10. (original) The method of claim 9, wherein the resolver determines a number of instructions that had already been executed using the mark instructions.

11. (original) The method of claim 10, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be changed, the resolver calls the copied instructions at the storage location so as to imitate a "no patch installed" scenario.

12. (original) The method of claim 11, wherein, after execution of the instructions at the storage location, the resolver returns control to the next instruction.

13. (original) The method of claim 1, further comprising enabling functionality of the copied instructions at the storage location.

14. (original) The method of claim 13, wherein the enabling step comprises reconciling addressing in the instructions in the storage location.

15. (original) The method of claim 13, wherein the enabling step comprises alignment of instructions in the instructions at the storage location.

16. (original) The method of claim 1, further comprising verifying that the original code is susceptible to patching.

17. (original) The method of claim 16, wherein the verifying step determines whether any mark instructions are already present in the original instructions.

18. (original) The method of claim 16, wherein the verifying step determines whether any copy protect instructions are already present in the original instructions.

19. (original) The method of claim 16, wherein the verifying step determines whether the original instructions include a suitable jump point that can be modified to the transfer of control to the hook.

20. (original) The method of claim 16, wherein the verifying step determines whether the original instructions represent valid instructions.

21. (original) The method of claim 1, further comprising placing the hook in memory.

22. (original) The method of claim 1, wherein the hook has been previously placed in memory.

23. (canceled)

24. (previously presented) The method of claim 1, wherein the atomic write replaces one instruction at a time.

25. (previously presented) The method of claim 1, wherein the atomic write replaces multiple instructions at a time.

26. (previously presented) The method of claim 1, wherein, for Intel IA-32 architecture, the atomic write uses any of "xchg", "lock cmpxchg8b," "lock cmpxchg," and "lock xchg" instructions.

27. (currently amended) A method of on-the-fly patching of executable code comprising:

verifying that original instructions to be patched are susceptible to patching while the original instructions are being executed on a processor;

generating pseudooriginal code by copying the original instructions to a different storage location from the original instructions;

~~adding a jump instruction to the pseudooriginal code to return to a next instruction after the original instructions~~ adding a hook with a first jump instruction ot the pseudoorignal code transferring control to the pseudoorignal instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, replacing the original code while the original instructions are in the process of being executed on the processor with tag instructions that indicate only their execution ~~and a transfer of control to a hook.~~; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the tag instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, and

wherein a number of times the tag instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions.

28. (original) The method of claim 27, wherein the original instructions are changed in reverse order.

29. (previously presented) The method of claim 27, wherein the tag instructions are the same length, in bytes, as the instructions to be patched.

30. (currently amended) The method of claim 27, wherein the tag instructions are shorter in length, in bytes, as the instructions to be patched, and ~~include~~ wherein NOP (no operation) filler follows the mark instructions.

31. (currently amended) A method of on-the-fly patching of executable code comprising:

verifying that original instructions to be patched are susceptible to patching while the original instructions are being executed on a processor;

generating pseudooriginal code by copying the original instructions to a different storage location from the original instructions;

~~adding a jump instruction to the pseudooriginal code to return to a next instruction after the original instructions~~ adding a hook with a first jump instruction transferring control to the pseudoorignal instructions by using a second jump instruction in the pseudoorignal instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

replacing the original code while the original instructions are in the process of being executed on the processor with tag instructions that indicate only their execution ~~and a transfer of control to a hook,~~;

calling the hook,

wherein the hook activates the second jump instruction to transfer control to the pseudoorignal instructions at a location just after the original instruction from which the hook was called,

wherein the tag instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the original instructions are part of the instruction set of the processor available to a user, and

wherein a number of times the tag instructions have been executed is ~~countable~~ counted in order to determine how many of the original instructions should be executed if control returns from an external execution context to the original instructions addresses after patching,

wherein the modified instructions include a resolver to determine a number of the instructions at a location of the original code that had already been executed.

32. (original) The method of claim 31, wherein the resolver determines a number of instructions that had already been executed using the tag instructions.

33. (currently amended) The method of claim 32, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be patched, the resolver calls the pseudooriginal code so as to imitate a "no patch installed" scenario.

34. (original) The method of claim 33, wherein, after execution of the pseudooriginal code, the resolver returns control to the next instruction.

35. (original) The method of claim 27, further comprising reconciling addressing in the instructions in the storage location.

36. (original) The method of claim 27, further comprising verifying that the original code is susceptible to patching.

37. (original) The method of claim 36, wherein the verifying step determines whether any tag instructions are already present in the original instructions.

38. (original) The method of claim 27, further comprising placing the hook in memory.

39. (canceled)

40. (currently amended) A method of on-the-fly patching of executable code comprising:

identifying original instructions to be patched while the original instructions are being executed on a processor;

allocating a storage location for storing a functionally equivalent copy of the original instructions;

copying the original instructions to the storage location;

adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook;~~,

calling the hook,

wherein the copied instructions include a first jump instruction to the pseudooriginal code and the hook includes a second jump instruction that returns operations to patched instructions after the original instructions,

wherein the hook activates the second jump instruction to transfer control to the pseudoorignal instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the original instructions are part of the instruction set of the processor available to a user, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions .

41. (original) The method of claim 40, further comprising, prior to the allocating step, allowing a write operation on a page in memory where the original instructions are located.

42. (canceled).

43. (original) The method of claim 40, wherein the original instructions are changed in reverse order.

44. (original) The method of claim 40, wherein the modified instructions include a resolver to determine a number of the instructions at a location of the original instructions that had already been executed.

45. (original) The method of claim 44, wherein the resolver determines a number of instructions that had already been executed using the mark instructions.

46. (previously presented) The method of claim 45, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be patched, the resolver calls the functionally equivalent copy so as to imitate a "no patch installed" scenario.

47. (original) The method of claim 46, wherein, after execution of the functionally equivalent copy, the resolver returns control to the next instruction.

48. (original) The method of claim 40, further comprising verifying that the original instructions are susceptible to patching.

49. (original) The method of claim 48, wherein the verifying step determines whether any mark instructions are already present in the original instructions.

50. (original) The method of claim 48, wherein the verifying step determines whether any copy protect instructions are already present in the original instructions.

51. (canceled)

52. (original) The method of claim 40, further comprising enabling functionality of the copied instructions at the storage location.

53. (currently amended) A computer useable removable storage unit having computer program logic stored thereon for executing on a processor, for on-the-fly patching of executable code, the computer program logic comprising:

computer program code means for identifying original instructions to be patched while the original instructions are being executed on a processor;

computer program code means for copying the original instructions to a storage location;

computer program code means for ~~adding a jump instruction to the copied instructions to return to a next instruction after the original instructions~~ adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, computer program code means for replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook,~~; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions.

54. (previously presented) The computer program logic of claim 53, wherein the original instructions are changed in reverse order.

55. (previously presented) The computer program logic of claim 53, wherein the mark instructions are the same length, in bytes, as the instructions to be patched.

56. (currently amended) The computer program logic of claim 53, wherein the mark instructions are shorter in length, in bytes, as the instructions to be patched, and ~~include~~ wherein NOP (no operation) filler follows the mark instructions.

57. (previously presented) The computer program logic of claim 53, wherein the hook includes a resolver to determine a number of the instructions at a location of the original code that had already been executed.

58. (canceled)

59. (previously presented) The computer program logic of claim 53, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be patched, the resolver calls the copied instructions at the storage location so as to imitate a "no patch installed" scenario.

60. (previously presented) The computer program logic of claim 59, wherein, after execution of the instructions to the storage location, the resolver returns control to the next instruction.

61. (previously presented) The computer program logic of claim 53, further comprising computer program code means for enabling functionality of the copied instructions at the storage location.

62. (previously presented) The computer program logic of claim 61, wherein the computer program code means for enabling functionality of the copied instructions at the storage location comprises computer program code means for reconciling addressing in the instructions in the storage location.

63. (previously presented) The computer program logic of claim 53, further comprising computer program code means for verifying that the original code is susceptible to patching.

64. (previously presented) The computer program logic of claim 63, wherein the computer program code means for verifying determines whether any mark instructions are already present in the original instructions.

65. (canceled)

66. (canceled)

67. (currently amended) A computer useable removable storage unit having computer program logic stored thereon for executing on a processor, for on-the-fly patching of executable code, the computer program logic comprising:

computer program code means for verifying that original instructions to be patched are susceptible to patching while the original instructions are being executed on a processor;

computer program code means for generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

computer program code means for ~~adding a jump instruction to the pseudooriginal code to return to a next instruction after the original instructions~~ adding a hook to the pseudooriginal code with a first jump instruction transferring control to the pseudoorignal instructions by using a second jump instruction in the pseudoorignal instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, computer program code means for replacing the original code while the original code is in the process of being executed on the processor with tag instructions that indicate only their execution ~~and a transfer of control to a hook,~~; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the pseudoorignal instructions at a location just after the original instruction from which the hook was called,

wherein the tag instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, and

wherein a number of times the tag instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions .

68. (currently amended) A computer useable removable storage unit having computer program logic stored thereon for executing on a processor, for on-the-fly patching of executable code, the computer program logic comprising:

computer program code means for identifying original instructions to be patched while the original instructions are executed on a processor;

computer program code means for allocating a storage location for storing a functionally equivalent copy of the original instructions;

computer program code means for copying the original instructions to the storage location;

computer program code means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

computer program code means for using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, computer program code means for replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook.~~; and

computer program code means for calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions.

69. (currently amended) A system for on-the-fly patching of executable code comprising:

means for identifying original instructions to be patched while the instructions are being executed on a processor;

means for copying the original instructions to a storage location;

means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched ~~adding a jump instruction to the copied instructions to return to a next instruction after the original instructions; and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, means for replacing the original code as while the code is in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook,~~; and

calling the hook

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions.

70. (currently amended) A system for on-the-fly patching of executable code comprising:

means for verifying that original instructions to be patched are susceptible to patching while the instructions are being executed on a processor;

means for generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

means for adding a hook with a first jump instruction to the pseudooriginal instructions transferring control to the pseudoorignal instructions by using a second jump instruction in the

pseudoorignal instructions that always transfers control to unpatched instructions in a location where the instructions are being patched ~~adding a jump instruction to the enabled pseudooriginal code to return to a next instruction after the original instructions;~~ ~~and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, means for replacing the original code while the original code is in the process of being executed on the processor with tag instructions that indicate only their execution ~~and a transfer of control to a hook,~~ and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the tag instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched;

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called, and

wherein a number of times the tag instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions .

71. (currently amended) A system for on-the-fly patching of executable code comprising:

means for identifying original instructions to be changed while the instructions are executed on a processor;

means for allocating a storage location for storing a functionally equivalent copy of the original instructions;

means for copying the original instructions to the storage location;

means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched; ~~and~~

using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all, means for replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions ~~and a transfer of control to a hook,~~; and

calling the hook

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called

wherein the original instructions are part of the instruction set of the processor available to a user, and

wherein a number of times the mark instructions have been executed is ~~countable~~ counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions.

72. (previously presented) The method of claim 1, wherein the process of execution of the original instructions is not interrupted throughout the patching process.

73. (previously presented) The method of claim 27, wherein the process of execution of the original code is not interrupted throughout the patching process.